

NAVAL POSTGRADUATE SCHOOL
Monterey, California

AD-A286 047



94-34418



THESIS

DTIC

EXACTE

NOV 09 1994



**A PERIODIC SCHEDULING HEURISTIC FOR
MAPPING ITERATIVE TASK GRAPHS ONTO
DISTRIBUTED MEMORY MULTIPROCESSORS**

by

Charles D. Kasinger

September 1994

Thesis Advisor:

Amr M. Zaky

Approved for public release; distribution is unlimited.

DTIC QUALITY INSPECTED 5

94 11 4 085

REPORT DOCUMENTATION PAGEForm Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503

1. AGENCY USE ONLY (Leave Blank)		2. REPORT DATE September 1994		3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE A Periodic Scheduling Heuristic for Mapping Iterative Task Graphs Onto Distributed Memory Multiprocessors (U)				5. FUNDING NUMBERS	
6. AUTHOR(S) Kasinger, Charles Darwin					
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-500				10. SPONSORING/ MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.					
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.				12b. DISTRIBUTION CODE A	
13. ABSTRACT (Maximum 200 words) This thesis investigates the problem of statically assigning the tasks of applications represented by repetitive task graphs (such as sonar or radar signal processing) to the processors of a distributed memory multiprocessor system with the objective of maximizing graph instance throughput. The repetitive nature of these task graphs allows for pipelining and the overlapping of successive graph instances, suggesting a departure from classical directed acyclic graph scheduling techniques. To investigate such a claim, a version of the Mapping Heuristic (MH) [ELR 90] is extended for use with iterative applications. Then a new heuristic, Periodic Scheduling (PS), is developed to capitalize on the repetitive nature of these task graphs by overlapping successive graph instances. The PS heuristic assigns tasks to processors in such a way so as to minimize the maximal utilization of the processors and the communications links between them. This maximal utilization figure dictates the interval between successive instances of the task graph. We conduct experiments in which the graph instance throughput of PS is compared to that of MH across a broad range of processor topologies, utilizing several communications/computation ratios. It is shown that, compared to MH, the PS heuristic improves the throughput performance between two and 50 percent. Particularly noteworthy improvement is noted on systems with high average inter-node communications costs.					
14. SUBJECT TERMS Assignment, distributed processors, heuristic algorithm, mapping problem, scheduling.				15. NUMBER OF PAGES 60	
				16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT Unlimited		

Approved for public release; distribution is unlimited

**A PERIODIC SCHEDULING HEURISTIC FOR MAPPING
ITERATIVE TASK GRAPHS ONTO
DISTRIBUTED MEMO MULTIPROCESSORS**

by

Charles D. Kasinger
Lieutenant Commander, United States Navy
B.B.A., Texas Tech University, 1983

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

September 1994

Author:

Charles D. Kasinger

Charles D. Kasinger

Approved By:

Amr Zaky

Amr M. Zaky, Thesis Advisor

Shridhar B. Shukla

Shridhar B. Shukla, Second Reader

Ted Lewis

Ted Lewis, Chairman,
Department of Computer Science

Decision For

DTIS	CR&I	<input checked="" type="checkbox"/>
ETIC	TAS	<input type="checkbox"/>
Unpublished		<input type="checkbox"/>

By _____

Date _____

Reviewed by _____

Approved _____

Special _____

A-1

ABSTRACT

This thesis investigates the problem of statically assigning the tasks of applications represented by repetitive task graphs (such as sonar or radar signal processing) to the processors of a distributed memory multiprocessor system with the objective of maximizing graph instance throughput. The repetitive nature of these task graphs allows for pipelining and the overlapping of successive graph instances, suggesting a departure from classical directed acyclic graph scheduling techniques. To investigate such a claim, a version of the Mapping Heuristic (MH) [ELR 90] is extended for use with iterative applications. Then a new heuristic, Periodic Scheduling (PS), is developed to capitalize on the repetitive nature of these task graphs by overlapping successive graph instances. The PS heuristic assigns tasks to processors in such a way so as to minimize the maximal utilization of the processors and the communications links between them. This maximal utilization figure dictates the interval between successive instances of the task graph. We conduct experiments in which the graph instance throughput of PS is compared to that of MH across a broad range of processor topologies, utilizing several communications/computation ratios. It is shown that, compared to MH, the PS heuristic improves the throughput performance between two and 50 percent. Particularly noteworthy improvement is noted on systems with high average inter-node communications costs.

TABLE OF CONTENTS

I.	INTRODUCTION	1
A.	OBJECTIVES	1
1.	Scope of the Thesis	3
B.	THESIS ORGANIZATION	3
II.	BACKGROUND	5
A.	THE MAPPING PROBLEM	5
1.	Processors	5
2.	Tasks	6
3.	Problem Definition	8
B.	PRIOR WORK	8
1.	Taxonomy of Task Allocation Methods	8
2.	Graph Partitioning	10
3.	Mapping	10
4.	Summary	13
III.	PERIODIC SCHEDULING	15
A.	REVOLVING CYLINDER ANALYSIS	15
B.	THE MAPPING HEURISTIC	16
C.	THE PS HEURISTIC	18
1.	Selecting a Processor	18
2.	Instance Overlap	24
IV.	RESULTS	29
A.	METHODOLOGY	29
B.	EXPERIMENTS	30
1.	Analysis by Topology	30
a.	Ring	31
b.	Hypercube	31
c.	Torus	32
d.	Fully Connected Network	33
e.	The average case	33
2.	Variation of Communications/Computation Ratios	35
3.	Buffering Requirements.	38
C.	GRAPH UNROLLING	40
1.	MH With Graph Unrolling	41
2.	PS With Graph Unrolling	43
V.	CONCLUDING REMARKS	47
A.	CONCLUSION	47
B.	FUTURE WORK	47
	LIST OF REFERENCES	49
	INITIAL DISTRIBUTION LIST	51

I. INTRODUCTION

Physical limits to computer hardware constrain the performance of traditional von Neumann architectures in compute-intensive applications. One such application of interest is Digital Signal Processing (DSP), such as with radar or sonar. As a result of the performance limitations of sequential architectures in executing these compute-intensive applications, multiprocessor systems have evolved as a cost-effective and powerful means of meeting the increasing demands of DSP and other similar problems.

Multiprocessor systems achieve high levels of performance by partitioning a program into tasks which can be executed concurrently, then simultaneously executing these tasks on separate processors. However, the efforts to exploit multiprocessor systems have been affected by issues concerning how to best utilize the resources of the system when it is executing a given application. Of prime importance among these issues is the mapping problem, or the problem of assigning the tasks of a parallel program to the processors of the system so as to achieve a desired performance characteristic.

Much research has been conducted into the development of methods to obtain suboptimal yet satisfactory assignments of tasks to processors in multiprocessor systems. These methods have taken a variety of forms, including graph-theoretic, mathematical programming, queuing theory, and search algorithms. Heuristics, another method of attacking the problem, involves the application of some algorithm in an attempt to obtain a satisfactory result.

A. OBJECTIVES

Certain applications, such as DSP, are iterative in nature, executing many times consecutively. In a DSP application data arrive periodically, resulting in the periodic invocation of the program. Repeating the same computation is an inherent characteristic of these applications. This characteristic allows for temporal concurrency or pipelining.

When pipelining takes place, a series of tasks are executed in stages. Each stage in the series is dependent upon the results obtained in earlier stages. These programs may be executed by overlapping successive iterations [HOA 93].

This thesis concentrates on the mapping problem with the objective of maximizing the throughput of iterative parallel programs executing on a distributed memory multiprocessor. This is contrasted to other research into the mapping problem, which has centered around the objective of minimizing the execution time of one instance of a parallel program.

A possible solution to the problem might be to assign a copy of the application's associated task graph to each processor in the system. In real world applications, however, the amount of software needed for such a technique would grow prohibitively large. Additionally, this method would create the need for a huge input/output bandwidth. Finally the response time of this technique would be unsatisfactory.

What may also appear to be a solution is the construction of a layered graph consisting of task execution and communications information. From this graph the path in which the heaviest edge has minimum weight is ascertained. Using a simple procedure to label the tasks, this method runs in $O(m^3n)$ time with m tasks and n processors. However, extending this technique to mapping iterative applications to distributed memory multiprocessors would require that the applications associated task graph possess certain characteristics which cannot be guaranteed. [BOK 88]

To pursue the objective of maximizing throughput in repetitive parallel programs, the Periodic Scheduling (PS) heuristic is introduced. The PS heuristic is a compile-time mapping heuristic. It assigns tasks represented as nodes in a directed acyclic graph to the processors of a distributed memory multiprocessor system of an arbitrary topology. The PS heuristic assigns tasks to processors in such a way so as to minimize the maximum utilization of processing and communications resources. In making these assignments, PS considers communications between tasks, resource contention and the topology of the multiprocessor system.

1. Scope of the Thesis

This thesis presents PS. Previous research into the problem of allocating tasks to processors in multiprocessor systems is explored. PS is implemented and experiments are conducted. A range of multiprocessor topologies is considered in the experiments, and various communications/computation ratios are used. The performance of PS is measured and compared with the performance of the Mapping Heuristic (MH) [ELR 90].

B. THESIS ORGANIZATION

Chapter II presents an overview of the mapping problem and highlights previous research into the problem. In Chapter III the PS heuristic is introduced and compared to the MH heuristic. Revolving Cylinder (RC) analysis is also discussed. In Chapter IV, research methodology is explained and the results of the experiments are analyzed. In Chapter V, conclusions are drawn from the research and suggestions for future work are given.

II. BACKGROUND

A. THE MAPPING PROBLEM

1. Processors

Parallel processing systems may be represented by use of an undirected connected graph. The vertices of this graph symbolize the processing elements of the system. The edges connecting the vertices symbolize the communications links between the processing elements. Figure 1 shows an example of a simple processor graph representing a parallel processing system consisting of four processors numbered 0 to 3.

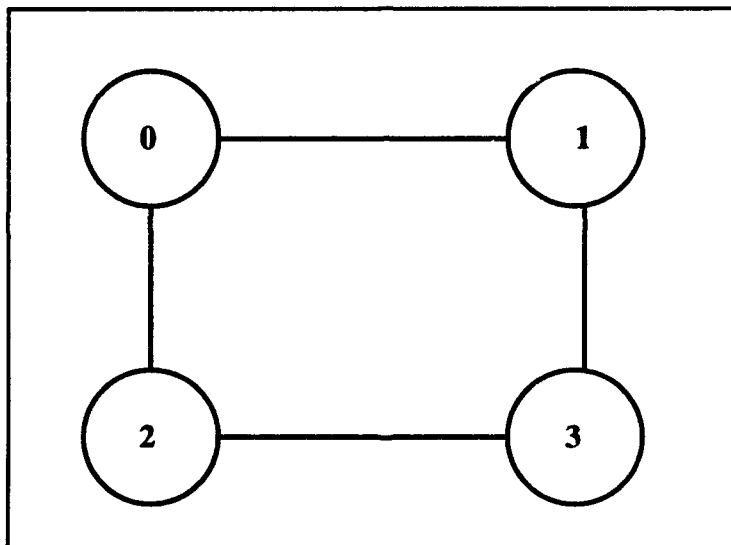


Figure 1: A simple processor graph.

We assume all processors are heterogeneous such that each processor executes the same task at an identical rate. Processors execute one task at a time. Simultaneous Interprocessor Communications (IPC) in either direction are possible. All communications links are assumed to be identical so that the cost of communications between tasks is independent of the processor assigned. There is no communications cost associated with

tasks assigned to the same processor, so that simultaneous task stop/next task start is possible on one processor.

2. Tasks

A task is defined as a portion of a parallel program that is executed sequentially. We consider tasks as discrete modules of instructions which communicate with their successors only upon termination of their computation. A task must finish before its successor starts executing, and a task cannot be preempted by a higher priority task.

Tasks of a parallel program and their interrelationships may be represented by a weighted directed acyclic graph known as a task graph. The tasks are symbolized by the vertices of the task graph. The edges between vertices indicate a precedence relationship and a communications event between two tasks. Specifically, the task at the tail of an edge must provide the results of its computation to the task at the head of the edge. Figure 2 shows an example of a simple task graph consisting of seven tasks. The letters (A-G) in the upper half of the nodes serve to identify the tasks. Numbers in the lower half of the nodes represent the amount of computation by that task. The numbers on the edges represent the amount of data to be communicated between the tasks.

Task granularity refers to the size of the individual task modules. When task partitioning takes place, a parallel program is broken down into modules of appropriate size. Adding tasks to a module increases the grain size of that module. In general, the larger the grain size, the less the parallelism and the smaller the grain size the more the communications. We assume that a task graph represents a parallel program which has been previously partitioned into modules of appropriate size.

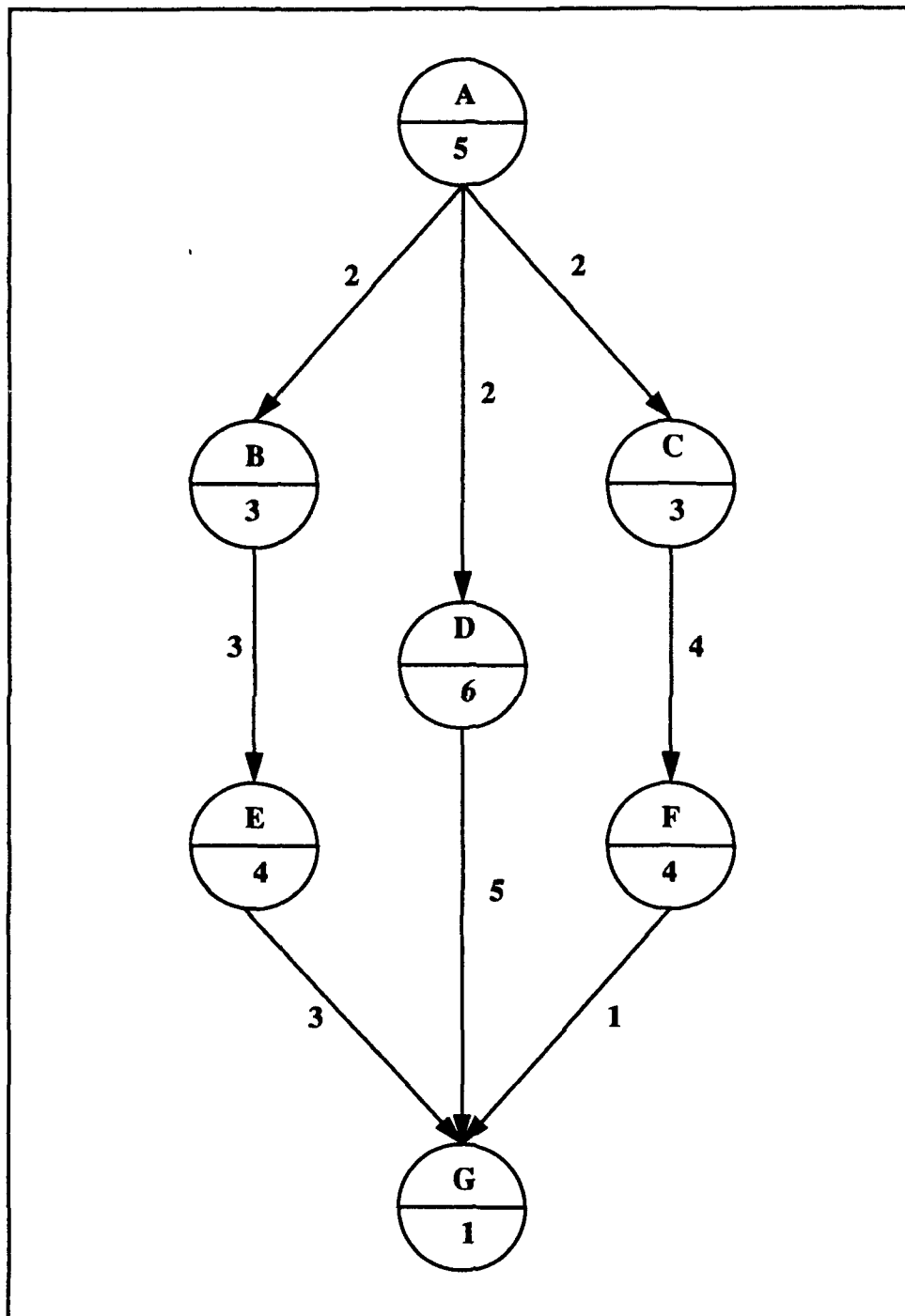


Figure 2: A simple task graph.

3. Problem Definition

The allocation of tasks to processors consists of two major components, mapping and scheduling. Mapping refers to the assignment of tasks to processors, while scheduling refers to the ordering of the execution of those tasks. We define the mapping problem as the problem of finding that arrangement which assigns tasks to processors so as to achieve some desired performance objective.

As noted previously, our objective is maximization of throughput, but the objective has traditionally been minimization of execution time. Execution time may be further decomposed into two parts: the time of computation and the time of communications. If all tasks are assigned to one processor then communications costs are minimized but computation time is maximized. Conversely, mapping an equal number of tasks to each processor would possibly maximize communications while perhaps minimizing computation.

B. PRIOR WORK

1. Taxonomy of Task Allocation Methods

The problem of allocating tasks to processors in multiprocessor systems has been widely studied. Casavant proposed a taxonomy of task scheduling techniques. According to this taxonomy scheduling techniques may be classified as static or dynamic. In static scheduling sufficient information is known concerning the processing elements within the multiprocessor system, as well as sufficient information concerning the tasks to be executed, prior to the execution of the program. Armed with such information beforehand, each executable may be statically assigned to a particular processor. In dynamic scheduling the decisions regarding the assignment of tasks to processors are made as the program executes. In these cases very little is known about tasks or processors prior to program execution. We shall be concerned with static scheduling. [CAS 88]

It has been shown by Ullman that the problem of finding optimal solutions to the mapping problem is NP-complete [ULL 75]. For this reason numerous methods have been proposed to yield suboptimal allocations which are satisfactory in terms of cost and results.

These suboptimal approaches can be divided into two categories. Approaches within the first of these two categories are known as approximate. Approximate solutions to the problem are obtained by ceasing to search for a solution as soon as a satisfactory one is found. This is contrasted to carrying out the search until an optimal solution is found. Effective approximate methods are dependent upon the availability of some metric for deciding when a solution is to be deemed satisfactory. [CAS 88]

Methods within the second suboptimal category are known as heuristics. Heuristic methods achieve their results while using less time and system resources than any of the other approaches. They arrive at their solutions by making certain assumptions concerning what is already known about the characteristics of the task graph and the multiprocessor system. Heuristics take advantage of certain conditions which exist within the system and which affect the system indirectly.

Heuristics may be classified in two ways: one pass-and iterative . One-pass heuristic algorithms make assignments of tasks to processors then do not change these assignments. Iterative approaches use an algorithm to attempt to improve on this initial assignment at least once. [HAM 92]

Within the categories of optimal approaches and approximate methods under static scheduling, four general categories are identified. Enumerative methods, the first category, list and search the solution space. Mathematical programming methods find solutions by means of integer programming techniques. Graph theoretic methods represent each task as a graph, then apply an algorithm to this graph to accomplish task assignment. The final category, queuing theoretic, applies algorithms based on queuing theory to the problem.

2. Graph Partitioning

Graph partitioning is related to graph mapping in that graph partitioning can be viewed as the first step of one method of obtaining satisfactory graph mappings. The graph partitioning problem consists of partitioning the nodes of a weighted graph into subsets of limited size so as to minimize the total cost of the edges which are cut.

An efficient graph partitioning heuristic is described by Kernighan and Lin. An algorithm for a 2-way partition of a graph of n vertices is presented. This algorithm finds an acceptable solution in $O(n^2)$ time. The technique was extended to multiple-way partitions via repeated application of the algorithm for 2-way partitions. It was found that two applications of the algorithm are usually sufficient to constitute more than 95 percent of the total improvement. [KER 70]

Zhou presented two task graph partitioning techniques for finding load-balanced task allocation or scheduling solutions. Based on clustering techniques, the approaches have a minimal execution time objective. It was noted that an inherent weakness of clustering techniques, a non-predictable number of clusters, can be solved using these approaches. [ZHO 93]

3. Mapping

Task allocation methods using network flow are introduced by Stone and Bokhari. Mathematical models were applied to representations of two-, three- and multi-processor systems. They showed that efficient solutions can be reached for systems of less than four processors using network flow approaches. This left open the question of the applicability of these approaches to more realistic multiprocessor systems. [STO 78]

A heuristic which permutes the adjacency matrix representing the task graph into a matrix which more closely resembles the adjacency matrix of the processor graph has been developed by Bokhari. The heuristic was designed for use on a specific target machine known as the Finite Element Machine, an array of processors. This heuristic

proved acceptable for 6 X 6 processor arrays but appeared to be less acceptable for very large arrays. [POK1 81]

Using a dynamic programming approach, Bokhari showed that the task allocation problem could be solved in $O(mn^2)$ time on a system of m communicating tasks and n processors, provided the pattern of communications among the tasks can be limited to be a tree. Programs having such a communications structure are of a common variety. This algorithm attempts to identify the shortest tree in a task graph in order to make task allocation assignment decisions accordingly. [BOK2 81]

An allocation approach based on a heuristic applied to a communication-ordered policy is presented by Evans and Kessler. The communication-ordered approach models task execution times and communications delays as simulation events. In this heuristic, global tasks are scheduled on idle processors, prioritized by the length of the longest precedence related path from the task to the end of the task graph. Local tasks are scheduled on available processors subject to certain restrictions based on communications costs and prioritized by these exit path lengths plus any communication delay saved. Finally, an attempt is made to schedule global tasks to the processors of their predecessors. In testing, this heuristic approach performed better than other algorithms with which it was compared across a range of communications cost ratios. Significant improvement was shown in highly parallel task graphs and with moderate to high communications costs. [EVA 92]

Also explored by Evans is the relationship between processor utilization induced by the allocation of tasks to processors and the quantity of processing elements which are available to partitions of a partitioned multiprocessor system. The utilization of processors was profiled for use as an input into multiprocessor partitioning. In simulation this method of task allocation was shown to be effective in minimizing program execution time. [EVA 92]

Graph theoretic, mathematical programming, and heuristic methods are combined by Chaudhary and Aggarwal. Using graphical representations of the task and of

the processor network, a third graph known as the "extended host graph" is produced. The task graph is then mapped onto this graph. The tasks are then allocated to the processing elements using two optimization procedures. [CHA 93]

A method of assigning tasks whereby graph matching with weak homomorphism between task graphs and processor graphs is pursued is presented by Shen and Tsai. The weak homomorphism chosen minimizes task turnaround time and is termed optimal weak homomorphism. This weak homomorphism is found by means of the A* algorithm noted in artificial intelligence. By minimizing task turnaround time, this method hopes to minimize IPC and optimize load balancing. [SHE 85]

A static mapping heuristic called declustering is developed by Sih and Lee. Clustering techniques divide the nodes of a task graph into groups of nodes. Each of these groups is then mapped to the multiprocessor system by some approach. The first part of the declustering algorithm is a clustering strategy using an analysis technique to compare the trade-off between parallelism and IPC. The algorithm establishes then decomposes the parallelism instances in order of importance. The granularity of the clustered nodes are then adopted to suit the particular multiprocessor system in use. [SIH 93]

A heuristic utilizing the branch and bound algorithm in the scheduling of cooperative, communicating nodes of a task graph is presented by Peng and Shin. The objective of their algorithm is to minimize the maximum task response time in real-time systems. The heuristic was found to be very efficient and extensible to problems with a variety of constraints. [PEN 93]

A number of heuristic algorithms are proposed by Lo. These algorithms have the goal of minimizing execution and communications costs resulting due to a static assignment of tasks to processing elements in a distributed system. Interference costs, which result when multiple tasks are assigned to the same processor, were included in these algorithms. These algorithms worked well in simulation on a variety of systems. Further, Lo's research indicated that extremely complex algorithms did not perform significantly better than more efficient algorithms. [LOV 88]

Gerasoulis and Yang study the trade-off between linear clustering and nonlinear clustering. In linear clustering tasks which have a successor-predecessor relationship are scheduled on the same processor, exploiting parallelism. In nonlinear clustering independent tasks are clustered together, reducing the parallelism. He finds linear clustering to be more suitable for coarse grain task graphs. Nonlinear clustering is preferred with finer grain tasks, since communications costs are lowered. [GER 93]

4. Summary

As can be seen from the previous research described here, many approaches to solving the mapping problem can provide satisfactory if not optimal results. Heuristics have been developed that take into consideration and make allowances for the contradictory objectives of minimizing IPC and maximizing parallelism. Not answered, however, are the effectiveness of these heuristics when the objective of mapping is changed from minimization of execution time to maximization of throughput in those applications represented by repetitive task graphs.

III. PERIODIC SCHEDULING

A. REVOLVING CYLINDER ANALYSIS

The Revolving Cylinder (RC) technique is a method for determining task node execution sequence in repetitive applications [SHU 92]. RC performs compile-time analysis of task graphs, producing a restructured graph as a product. Although the RC method was designed for use in the assignment of task nodes on a specific shared memory multiprocessor, the theory of the technique is extensible to this research involving static mapping in a distributed memory multiprocessor system.

The cylinder is a logical data structure used to ascertain if a task graph may be mapped to a target multiprocessor system given requisite data rates. If the schedule of a task graph were wrapped around, such that its end touched its beginning, the effect would be to create a cylinder. To produce the cylinder the execution time of all task nodes are summed. The resulting figure is divided by the number of processors. This gives the cylinder's circumference, which is the maximum throughput that the target system will support, since each processor is utilized fully.

Each processor in the system is assigned one band of the cylinder. Each of these bands can be thought of as an equal portion of the total computation required to complete the task graph. Each of these bands is divided into slots. These slots are equal in size to the size of the smallest node in the task graph.

Consider the case of an application (such as DSP) where data arrives periodically. This results in the periodic instantiation of the application's associated task graph. When the task graph is mapped using the RC approach, the part of a cylinder correlating to a specific node is equal to the execution time of that node. Each cylinder is a schedule for one instance of the graph. Subsequent instances of the task graph are overlapped with the

first. Each node is assigned an index to ensure there are no conflicts between instances of the task graph.

To illustrate RC scheduling, the sample task graph shown in Figure 3 (without communications costs along its edges) is subjected to the RC technique. Figure 4 graphically depicts a possible result of RC scheduling applied to the task graph on a two processor system.

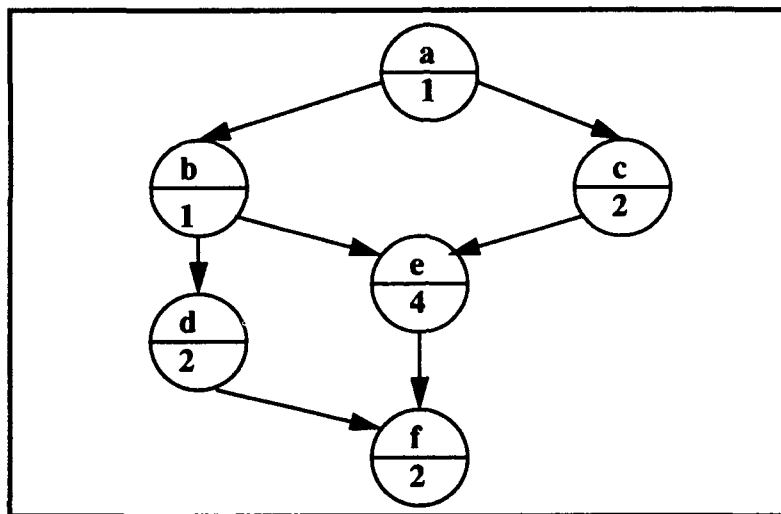


Figure 3: Example task graph [AKI 93].

B. THE MAPPING HEURISTIC

The Mapping Heuristic (MH) schedules tasks to distributed memory multiprocessors while taking into consideration such issues as processing system topology, communications, contention for resources, and the trade-off between IPC and computation. [ELR 90]

MH is an adaptation of a class of scheduling heuristics known as list scheduling. List schedules assign priorities to tasks and place the tasks on a list in decreasing order of priority. Higher priority tasks are assigned to processors first. Once a task is ready to run, MH assigns that task to a processor such that the task cannot finish earlier on any other processor. In making this decision, MH considers the processor topology, contention for resources and the speed of the processors.

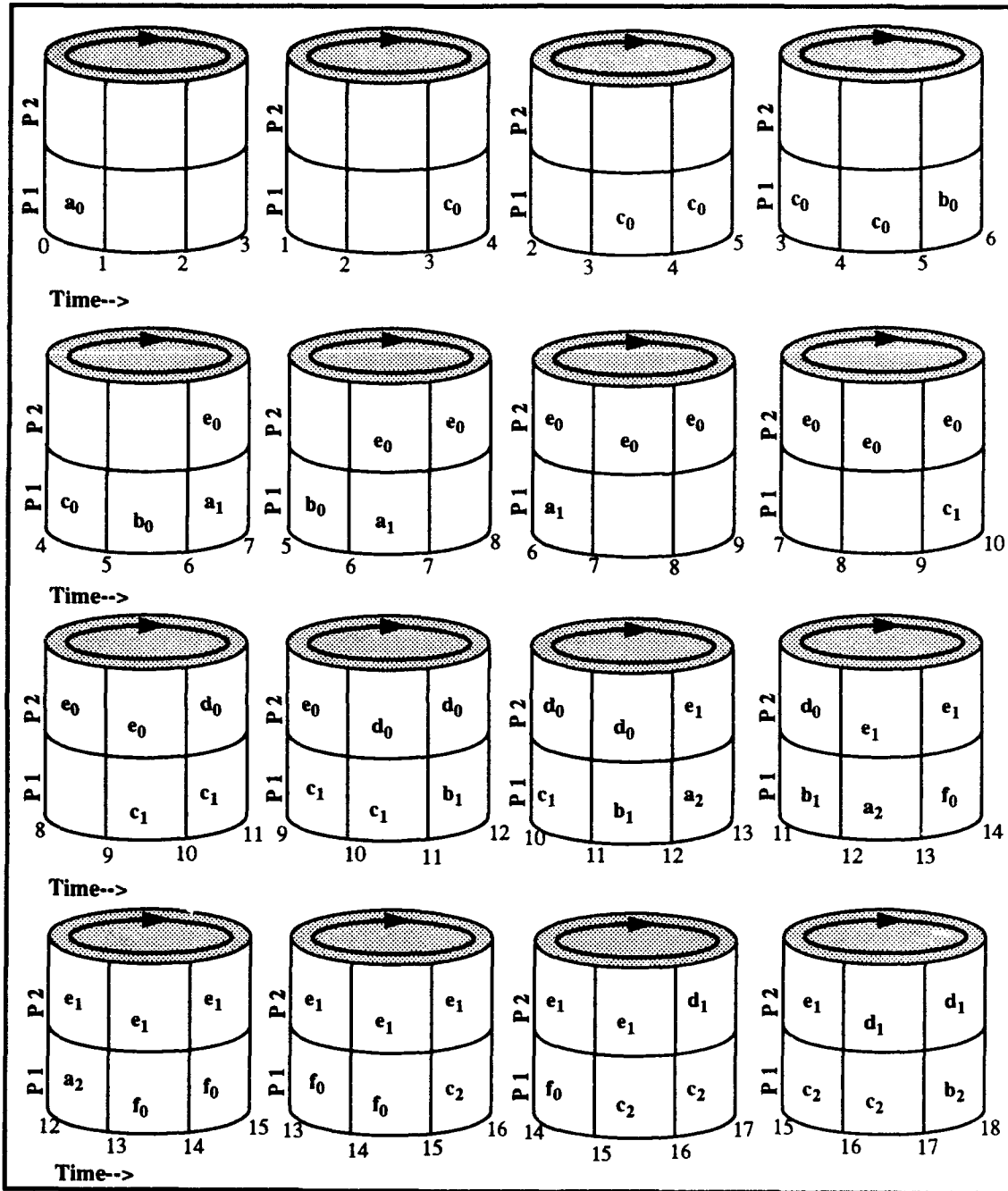


Figure 4: Revolving cylinder applied to the example task graph in Figure 3 [AKI 93].

Delay between the moment a task finishes on one processor and the moment a successor task commences on a second processor consists of two parts. The first part is the time actually spent transmitting data over the link. To this is added the time spent waiting for other messages sent over the same route, if any, to clear the route. This second part is contention delay.

To keep track of contention, MH maintains routing tables indexed by the number of processors. The (x,y) cell of the routing table maintains information concerning current contention delay, the number of hops and the preferred outgoing line from processor x to processor y . During the scheduling process, routing tables are updated to reflect the most current communications contention information.

There is a trade-off between the frequency of updating and the expense of updating. MH updates the routing tables on the occasion of one of two events: (1) a task on one processor begins transmitting a message to a task on another processor, adding contention to links between these processors, or (2) a message arrives at the destination processor, removing contention from the links along the route between the processors. When the example task graph shown in Figure 5 is mapped by MH to a four processor ring, the result is represented by the Gantt chart shown in Figure 6. The labelled cells in Figure 6 indicate periods of task execution, with the label representing the specific task being executed. Shaded areas are periods that the specified processors (P1-P4) are idle. The total execution time of one instance of the graph is 12 units.

C. THE PS HEURISTIC

1. Selecting a Processor

PS is similar to MH in several ways. PS considers communications, contention, and the interconnection network in assigning tasks to processors. PS makes use of routing tables to maintain current information on contention, number of hops and preferred outgoing line between processors. Like MH, any processor system topology is supported by PS, as is any processor speed.

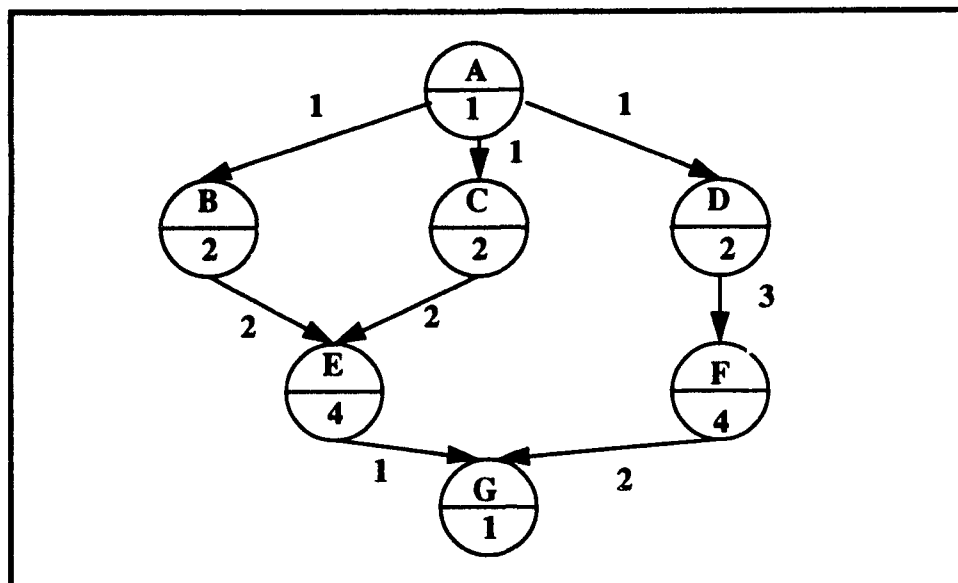


Figure 5: Task graph for mapping by MH and PS.

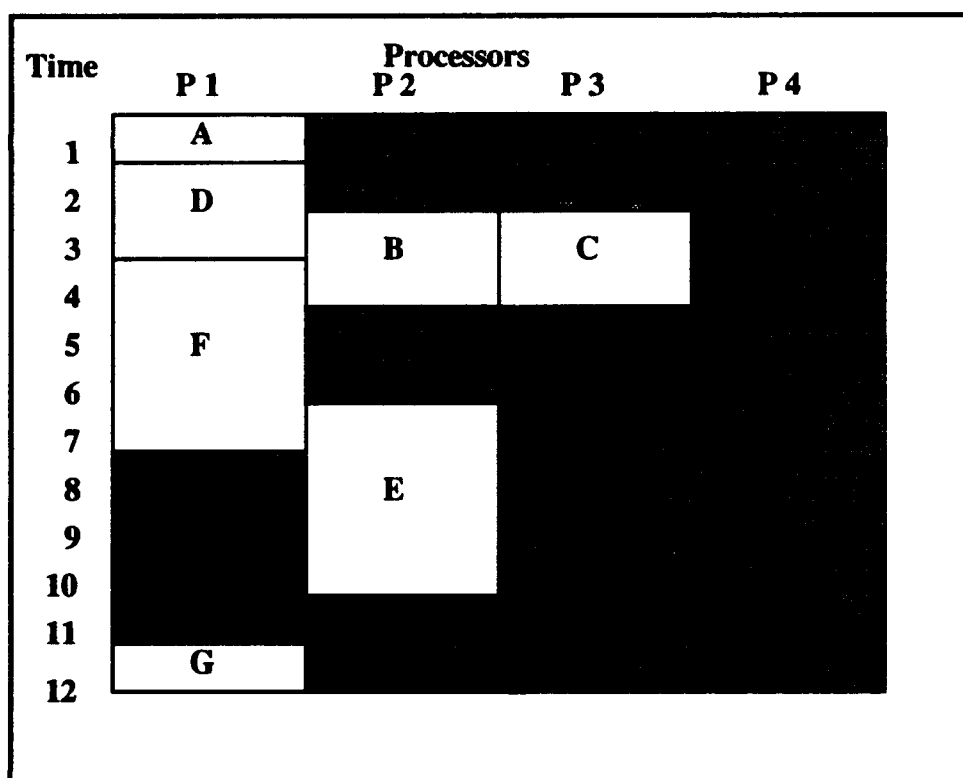


Figure 6: MH mapping of task graph given in Figure 5.

PS differs from MH in some key respects. The goal of MH is to minimize the execution time of a given task graph. The goal of PS is to maximize the throughput of a repetitive task graph. PS differs from MH, also, in the heuristic by which a processor is selected to run a given task. PS selects the processor which minimizes the maximum utilization of resources. Those resources include processing elements and the communications links which connect those elements. Towards this end, PS maintains a Processor Utilization Table to hold information on the level of utilization of the processors and a Link Utilization Table to hold information on the level of utilization of the communications links in the interconnection network.

Figure 7 gives the pseudocode for the PS algorithm which chooses a processor on which to run a task. When a task is ready to run, it is passed to the procedure, along with the Processor Utilization Table and the Link Utilization Table. Copies of the utilization tables are made for testing purposes within the procedure.

The ready task is tested on each processor in the system. The copy of the Processor Utilization Table is changed to reflect how an assignment to a given processor will affect the utilization level of that processor.

The ready task may have predecessors which are assigned to other processors. If so, the copy of the link utilization tables is changed to reflect the effect on the links of the mapping of the ready task to a given processor. After all predecessors have been taken into account, the copies of the resource utilization tables accurately reflect the effects of assigning the ready task to the processor under question. The table copies are now searched to find the maximum utilization of any resource, and that information is stored. After each processor has been tested, the processor that results in the minimum of these maximal utilization levels is chosen as the processor on which to run the task. In mapping the task graph in Figure 5 to a four processor ring, PS would first examine the impact on resource utilization of assigning task A to processor 1. Figure 8 shows the results on the link and processor utilization tables of such a mapping.

```

procedure Select_Processor (Ready Task,
                             Processor Utilization Table,
                             Link Utilization Table)

for each Processor in the system

    increment the associated Processor Utilization Table by
    the computation time of the Ready Task

    if the Ready Task has predecessors then

        for each Predecessor of the Ready Task

            if the Predecessor is not assigned to this Processor then

                increment the Link Utilization Table of the links
                connecting this Processor to the processor on
                which the predecessor executed by the amount of
                data sent from the Predecessor to the Ready Task

            end(if)

        end(for)

    end(if)

    find the maximum of the utilization figures for any link or
    processor, given the assignment of the Ready Task to this Processor

    if this maximum utilization figure is less than the maximum
    utilization figure given the assignment to any other
    processor then

        map the Ready Task to this Processor

    end(if)

end(for)
end.

```

Figure 7: Algorithm used by PS to select a processor on which to execute a task.

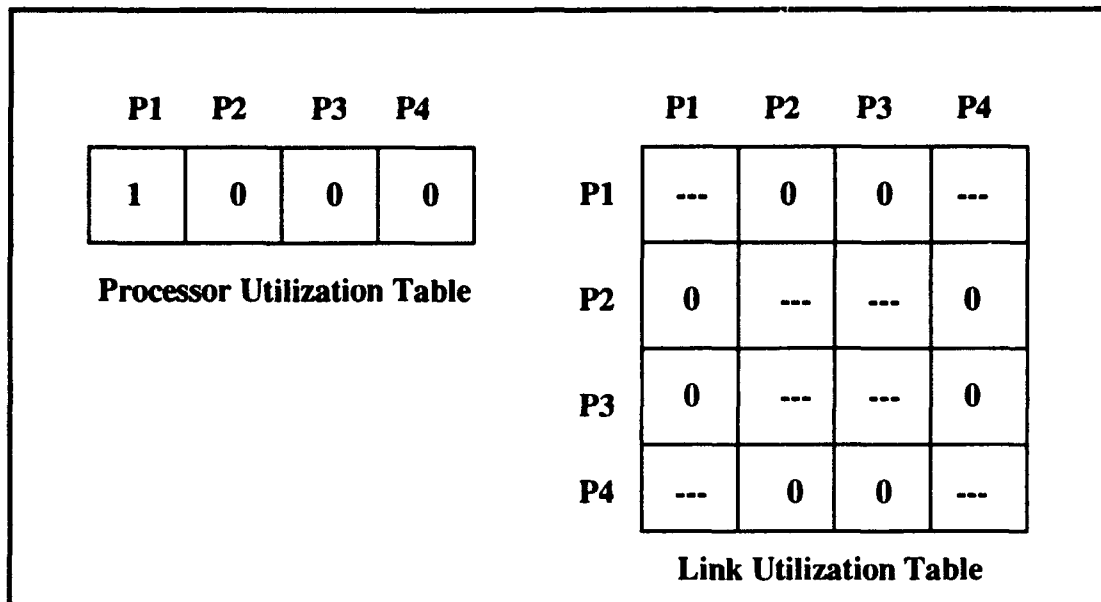


Figure 8: PS Processor and Link Utilization Tables for assignment of task A to processor 1.

Similar results ensue when the task is examined on other processors. PS selects the first "best" mapping, assigning task A to processor 1.

Next PS examines the impact of assigning task B to each processor. Placing task B on processor 1 would result in the situation depicted in Figure 9, with a maximal utilization of three on processor 1. If task B is placed on any processor other than processor 1, the data produced by task A on processor 1 must be forwarded to the chosen processor. Figure 10 depicts the potential impact of assigning task B to processor 2. This would give processor 2 a utilization level of two and would increase the utilization of the link between processor 1 and processor 2 to a level of 1. Assigning task B to processor 3 would similarly effect resource utilization. An assignment to processor 4 might result in the resource utilization levels depicted in Figure 11. Data transmitted from task A on processor 4 would necessarily transit two links (1-2 and 2-4 in this case). Having tested all four possibilities, PS assigns Task B to processor 2.

After every assignment involving the links between processors, the effects of the assignment with respect to link contention are examined and preferred interprocessor

routes are updated to reflect the contention. PS utilizes the same methodology for this purpose as does MH [ELR 90].

P1	P2	P3	P4
3	0	0	0

Processor Utilization Table

	P1	P2	P3	P4
P1	---	0	0	---
P2	0	---	---	0
P3	0	---	---	0
P4	---	0	0	---

Link Utilization Table

Figure 9: PS Processor and Link Utilization Tables for assignment of task B to processor 1.

P1	P2	P3	P4
1	2	0	0

Processor Utilization Table

	P1	P2	P3	P4
P1	---	1	0	---
P2	0	---	---	0
P3	0	---	---	0
P4	---	0	0	---

Link Utilization Table

Figure 10: PS Processor and Link Utilization Tables for assignment of task B to processor 2.

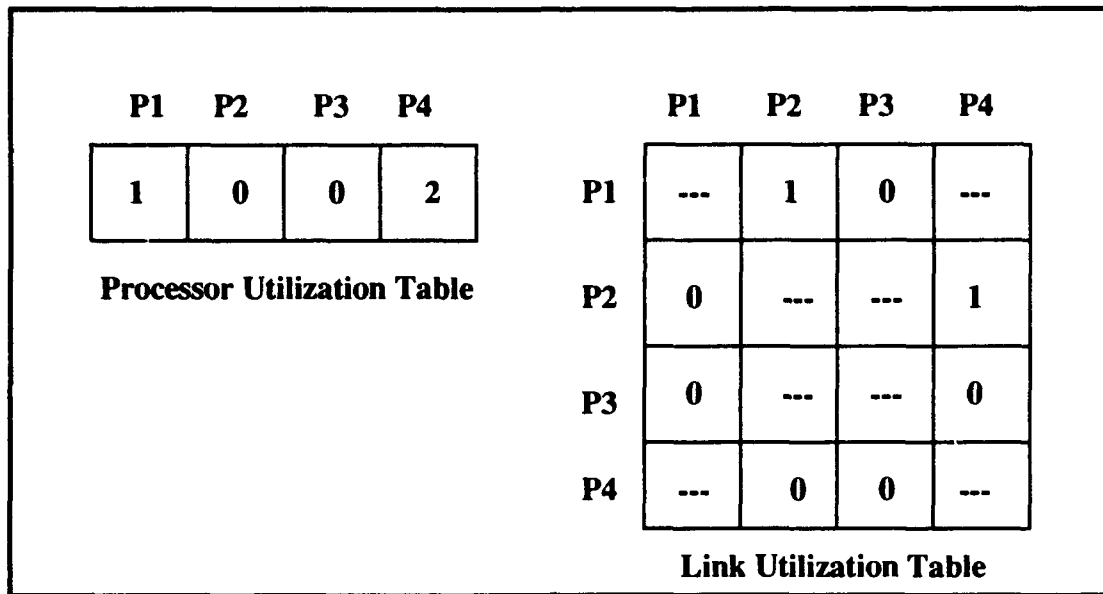


Figure 11: PS Processor and Link Utilization Tables for assignment of task B to processor 4.

As each task is ready to run, the above procedure is repeated. The potential impact on system resources of assigning a task to a processor is examined for each processor in turn. That processor is chosen that minimizes the maximal utilization of all resources.

2. Instance Overlap

An essential concept of periodic scheduling is its application to iterative task graphs. As the term implies, iterative task graphs represent those applications which execute many times consecutively. DSP applications are an example. Large amounts of data are continuously input and output by DSP applications on a real time basis. These applications are characterized by the repetition inherent in their associated task graphs. This repetition allows for the RC property of overlapping task graphs. Overlapping task graphs represent the fact that at any point in time portions of more than one instance of a task graph may be actively undergoing computation.

Once PS determines to which processor a task will be assigned, that task is assigned at the earliest time that the processor comes available. In other words, no consideration is given at this time to task precedence relationships. If a precedence relationship exists between the task and another task, then that task is assumed to be part of another instance of the task graph which is overlapped with other currently executing tasks. Since that task cannot be executed until its parents have executed, producing the data necessary for its computation, analysis of the assigned tasks is essential in order to determine to which instance of a task graph the task actually belongs.

To analyze the precedence relationships between tasks mapped in overlapping task graphs, the method used in RC scheduling is implemented [LIT 91]. This algorithm assigns indices to the allocated tasks to represent to which instance of a graph the task belongs. The algorithm compares the completion time of a task with its predecessors and successors to establish which index should be assigned to that task. PS uses a modified version of the algorithm to include the effects of contention and communications costs. Figure 12 is a Gantt chart showing how the mapping of the task graph of Figure 5 to a four processor ring using PS might look if instance overlap were not accomplished. Figure 13 shows the same mapping, with instance overlap and the addition of indices representing the instances associated with each task.

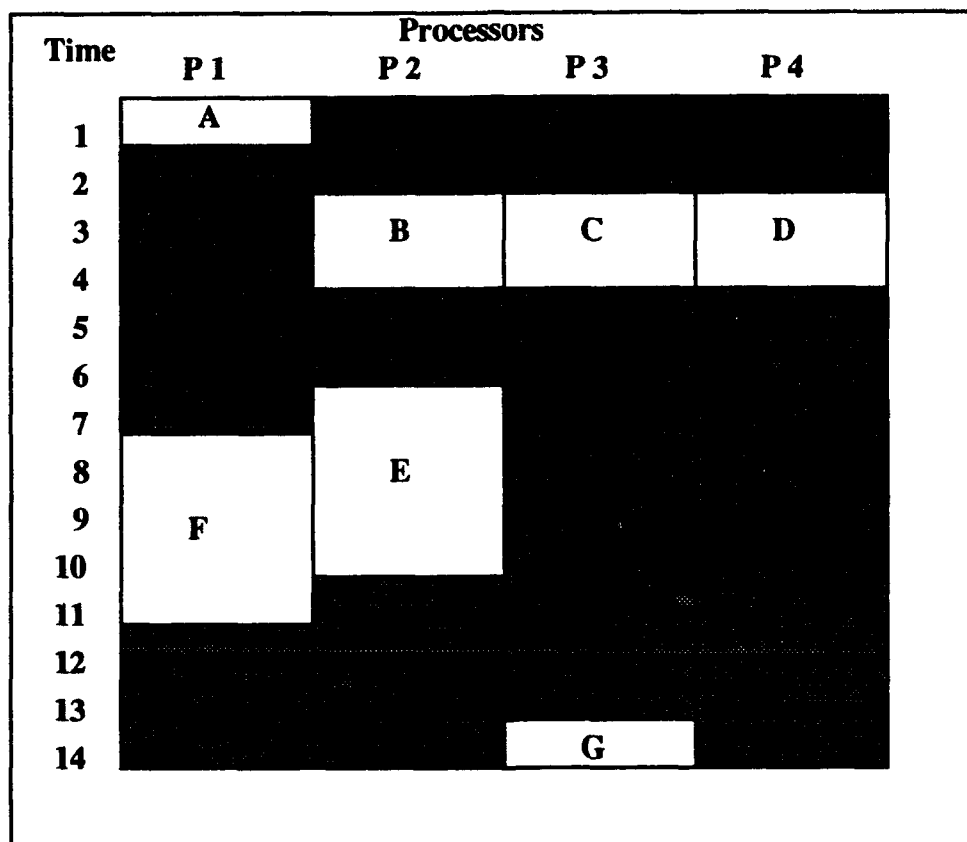


Figure 12: Mapping of Figure 4 task graph to a four processor ring without instance overlap.

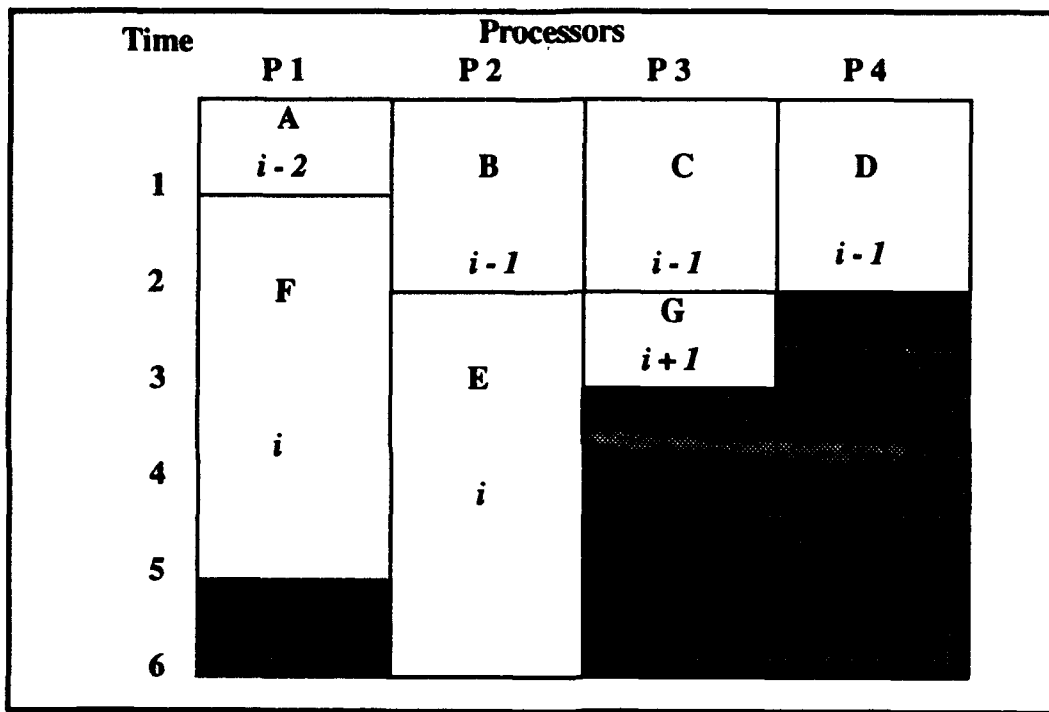


Figure 13: Mapping of Figure 4 task graph under PS with instance overlap.

IV. RESULTS

A. METHODOLOGY

Experiments were conducted to ascertain the throughput potential of the PS heuristic. The results of PS are compared with the results of MH. Although the MH heuristic is not specifically designed for mapping iterative task graphs, it may be extended to that purpose by repeating its application to repetitive copies of a task graph.

To conduct the experiments, 250 random task graphs were generated. Each of these task graphs consists of 60 nodes and 100 edges. The task nodes in each graph have an average execution time of 10,000 units, giving the graph an average aggregate of 600,000 units. These 250 graphs were subdivided into five groups of 50 graphs. Each of these groups consists of graphs averaging a particular communications/computation ratio. Ratios used in the research were 0.01, 0.1, 0.3, 0.7, and 1.0. These ratios represent a spectrum ranging from task graphs with negligible communications (i.e. computation intensive applications) to those with equal computations and communications.

Four topologies are considered. The ring topology represents an interconnection network with severely restricted communications flow. The fully connected network is included to represent the least restrictive network. In between these two extremes, we include the hypercube and the torus (toroidal mesh). Within each of these four topologies, we consider systems consisting of four, eight and sixteen processors.

It should be noted that this cross-section of topologies and processor totals does result in some duplication of target multiprocessing systems. For example, with four processors the torus, the ring and the hypercube represent the same interconnection network. Likewise, the torus and the hypercube are identical when the system consists of eight processors.

Since the objective of PS is to maximize throughput we wish to start a new instance of the task graph as often as possible. Towards this end, the performance of a heuristic is measured by the heuristic's normalized execution time. The normalized execution time of a heuristic is arrived at by dividing the actual finish time of a task graph when mapped by that heuristic by the lower bound on the execution time of that task graph. The lower bound on the execution time of a task graph is arrived at by dividing the completion time of one instance of the task graph on a sequential processor (600,000 units of time, on the average, for the graphs used in this research) by the number of processors in the target system. The lower bound for these experiments, then, is 150,000, 75,000 or 37,500 time units for systems of four, eight or sixteen processors, respectively.

The normalized execution time of a heuristic will be measured along the y-axis of subsequent graphs in this section. It should be noted that numbers along the y-axis differ throughout the various figures.

B. EXPERIMENTS

1. Analysis by Topology

First we consider the performance of PS in executing one instance of a task graph. PS and MH are used to map task graphs onto the same target multiprocessor systems, and the results are compared. The communications/computation ratio is limited to 0.3 in the experiment, representing a mid-range communications load. Analysis of results given four processors, eight processors and sixteen processors will be undertaken for each topology. The average case will also be considered.

In all cases, a distinct advantage is noted when the PS heuristic is used for mapping to systems composed of a greater number of processors. On systems of fewer processors, tasks are less likely to be mapped until their predecessors have executed. As the number of available processors increases, tasks generally are available for mapping sooner. Under MH, however, precedence relationships result in delays in executing these tasks until predecessors have completed. Consequently, more processor idle time, less

efficiency and a longer execution time results. Under PS, the task is mapped and packed and precedence relationships are handled by node indexing. Better results ensue.

a. Ring

The ring is the most restrictive topology studied in terms of communications flexibility. It is also the topology which shows the greatest difference in performance of the two heuristics. As seen in Figure 14, the normalized execution time of PS and MH are similar on rings of four processors. With eight processors the difference becomes somewhat more significant. The most dramatic difference is evident on systems of sixteen processors. On these systems the average normalized execution time of one instance of the graph under PS constitutes about a 60 percent improvement over that using MH.

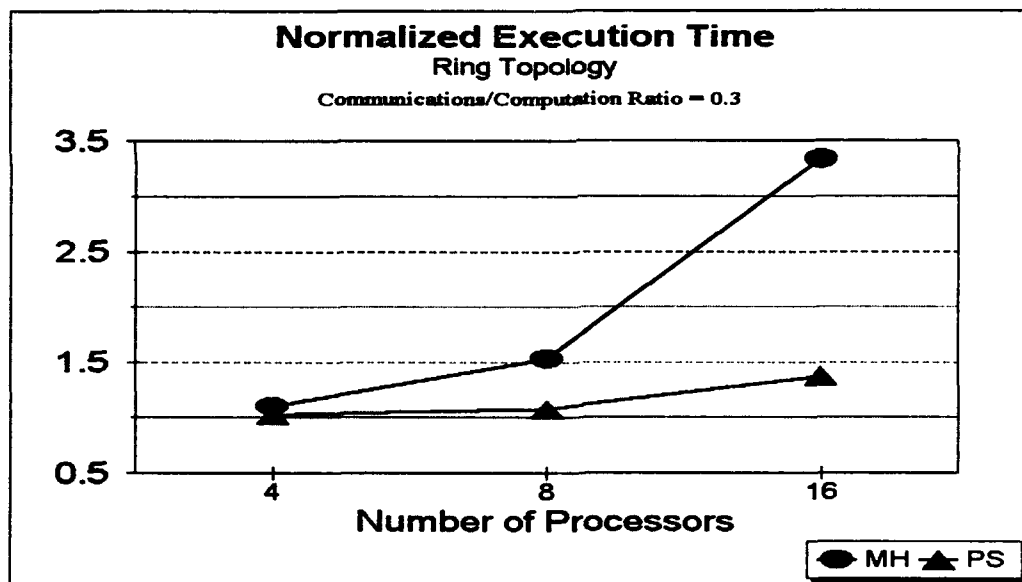


Figure 14: Normalized execution time of PS and MH on a ring of processors.

b. Hypercube

Mapping one instance of a task graph under MH on a hypercube of four processors shows only a slight disadvantage compared to a mapping produced by PS, as

shown in Figure 15. On systems of eight processors, the results under PS show a more marked advantage. When mapped to a sixteen processor hypercube, however, the most pronounced difference in performance is observed. A mapping by PS executes in about half the time it takes the mapping produced by MH.

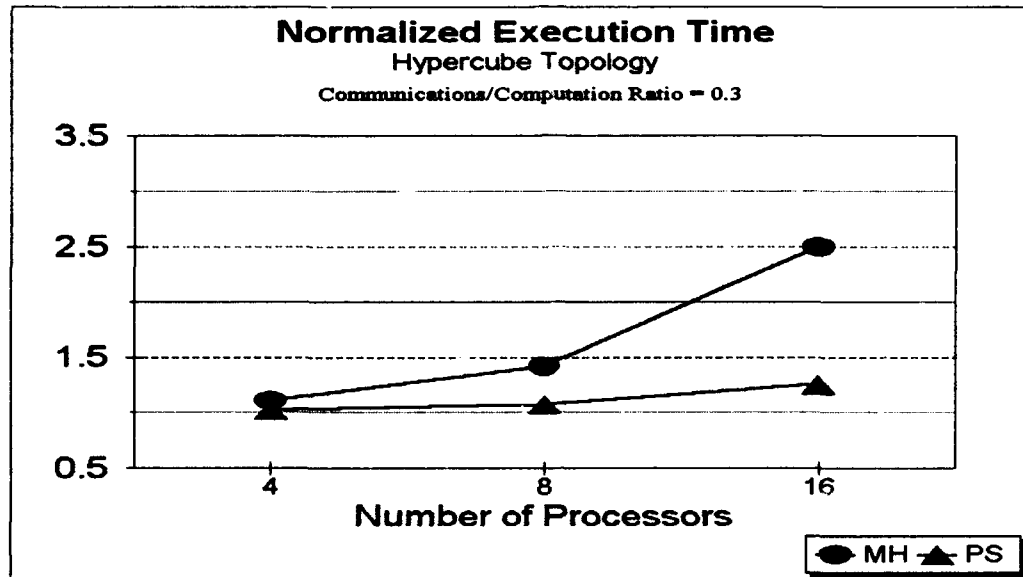


Figure 15: Normalized execution time of PS and MH on the hypercube.

c. Torus

Figure 16 depicts the results of tests conducted on the torus topology. On torus systems the difference in performance between PS and MH is virtually identical to that seen on the hypercube topology. Specifically, little difference appears on four processor systems. A more distinct advantage of PS is evident on eight processor systems. On sixteen processor systems the PS mapping is roughly twice as fast as a mapping by MH.

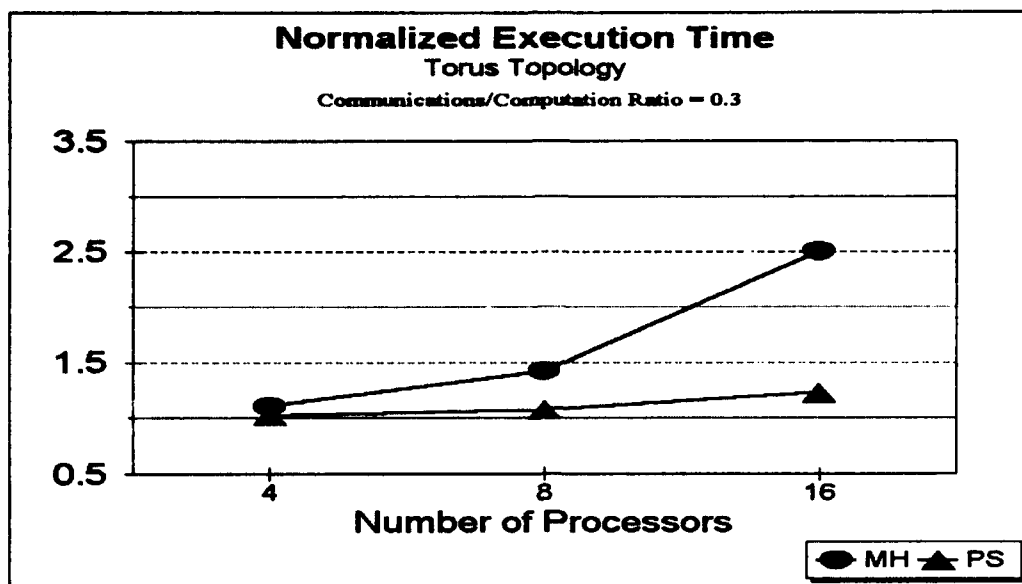


Figure 16: Normalized execution time of PS and MH on the torus.

d. Fully Connected Network

Figure 17 shows the efficiency of the two heuristics on the fully connected network. Normalized execution times are very similar to those of the torus and the hypercube. On a fully connected network of four processors, PS and MH perform similarly. The advantage of PS grows as the number of processors grows. On the average, on sixteen processor systems, mappings produced by PS result in a normalized execution time about half that of mappings produced by the MH heuristic.

e. The average case

Figure 18 depicts the average performance of PS and MH across all topologies. In general, PS shows the greatest advantage over MH in systems with a larger number of processors. Even on systems of fewer processors, however, PS still gives some advantage over MH.

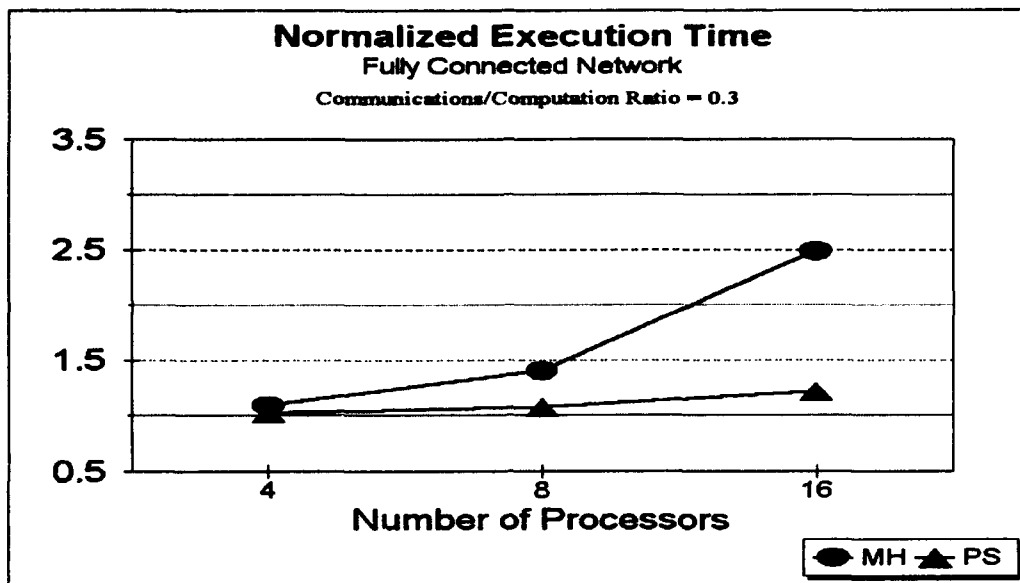


Figure 17: Normalized execution time of PS and MH on fully connected network.

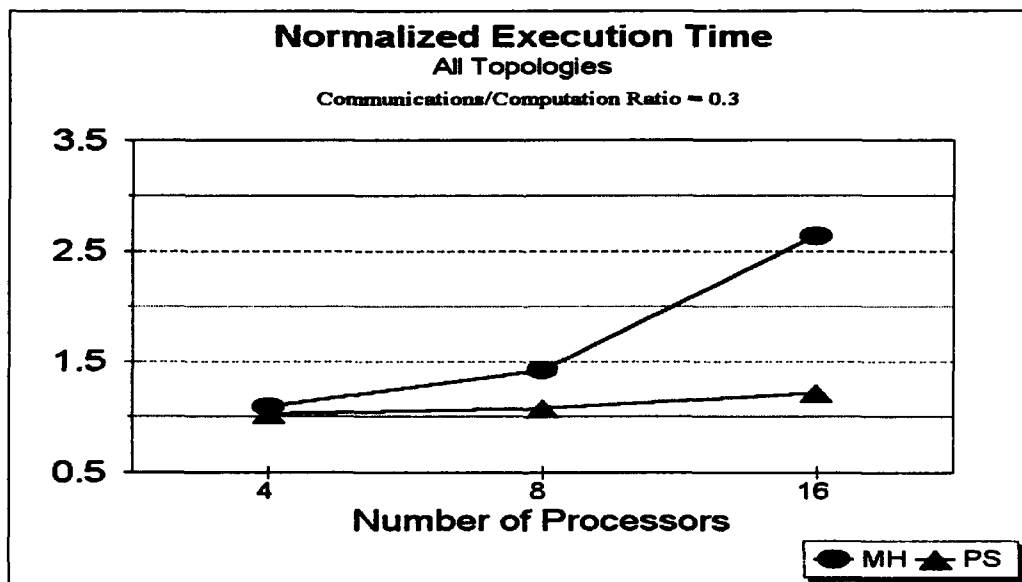


Figure 18: Normalized execution time of PS and MH across all topologies.

2. Variation of Communications/Computation Ratios

Next, attention is turned to the effects on the two heuristics of varying the communications/computation ratios within the task graphs. Only systems composed of sixteen processors are considered. All topologies are tested.

Figure 19 shows the results on the ring topology. On topologies with restricted communications flow, and as the amount of communications in an application grows, execution under the PS heuristic begins to slow. As the cost of utilizing a communications resource becomes higher, PS will choose to assign a task node to the same processor as its predecessor, as this serves to minimize the utilization of the communications resources. This results, however, in a lengthier execution time, as seen by the increase observed in Figure 19 at communications/computation ratios beyond 0.3. However, the increases also effect the MH heuristic, and PS still maintains a distinct performance advantage over MH at communications/computation ratios ranging above 0.3.

On the hypercube and torus topologies, shown in Figure 20 and Figure 21, respectively, a similar drop-off in performance is noted at communications/computation ratios beyond the mid-range value. Although PS still has an advantage in normalized execution time at all levels, the advantage is less marked.

Figure 22 shows the comparison of normalized execution time with varied communications/computation ratios on the fully connected network. The decline in advantage of PS noted with other topologies is not evident on the fully connected network, reflective of the greater communications flexibility of the topology.

In Figure 23, the normalized execution time of the two heuristics is presented as an average of all topologies. At all levels, PS maintains a performance advantage over the MH heuristic.

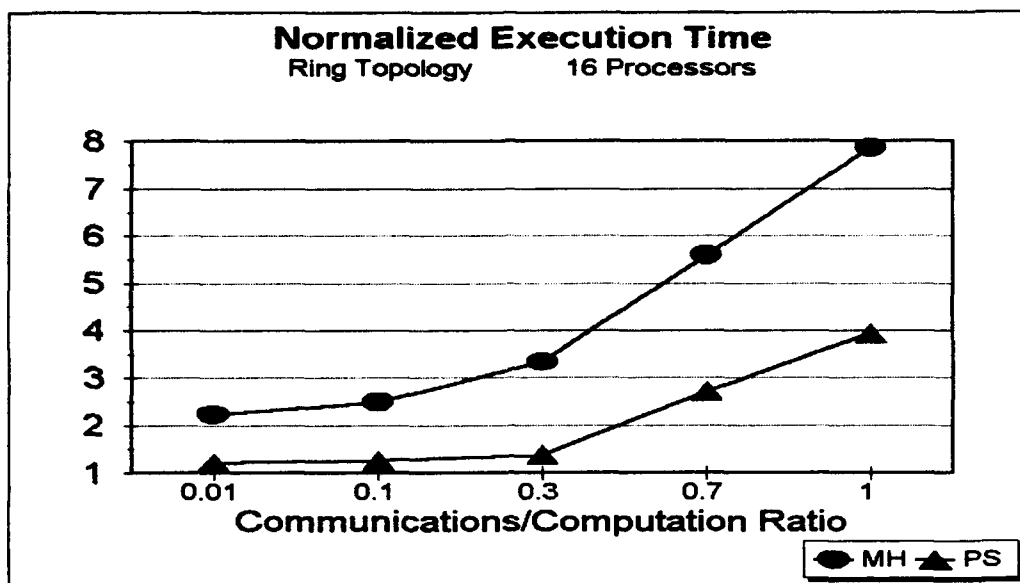


Figure 19: Normalized execution time on the processor ring when communications/computation is varied.

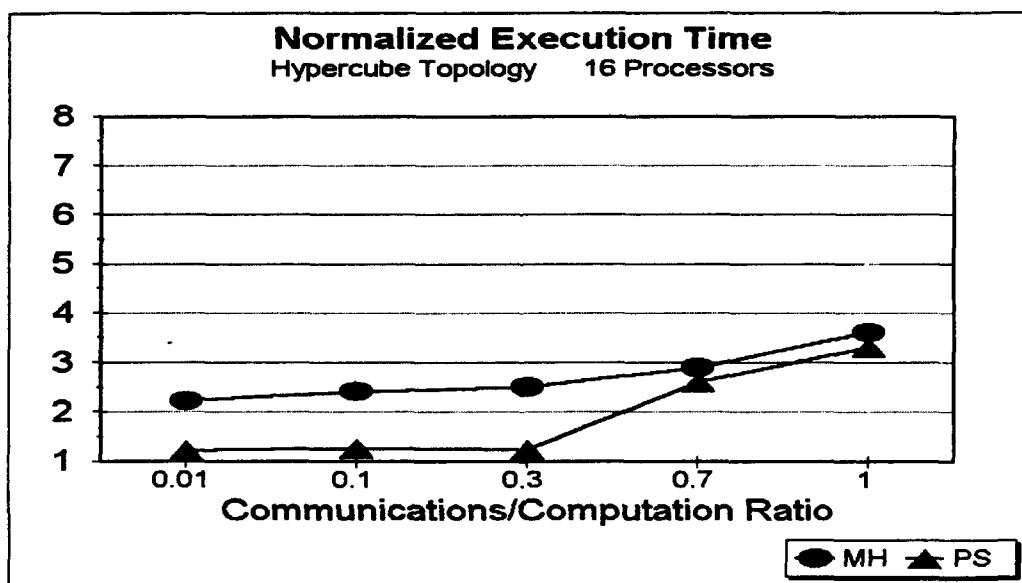


Figure 20: Normalized execution time on the hypercube when communications/computation is varied.

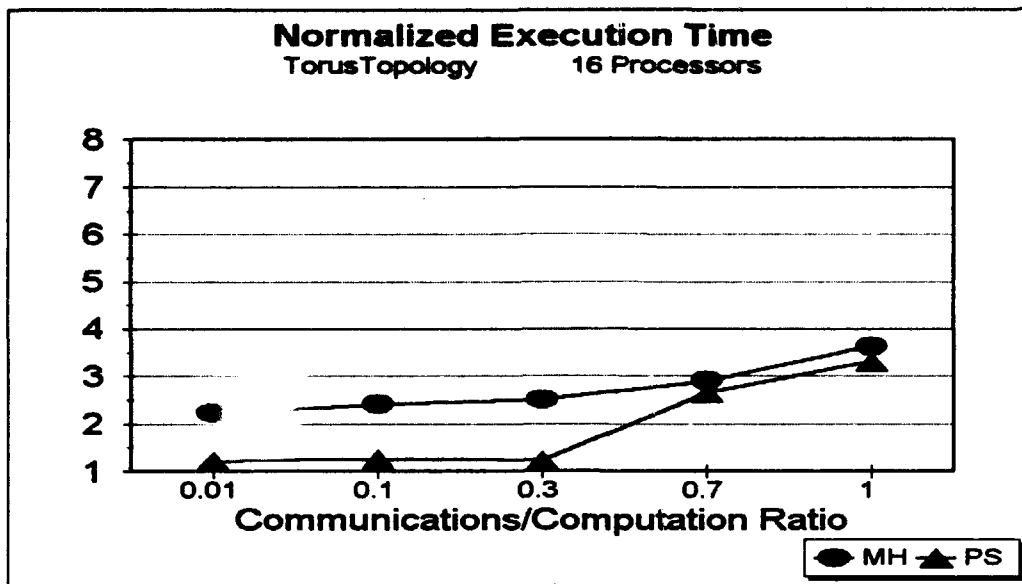


Figure 21: Normalized execution time on the torus when communications/computation is varied.

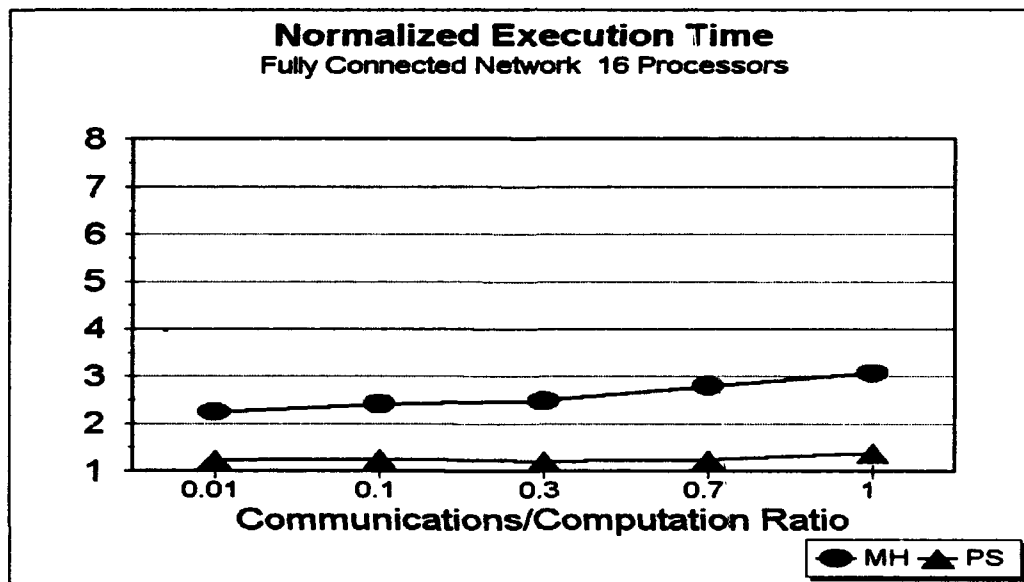


Figure 22: Normalized execution time on the fully connected network when communications/computation is varied.

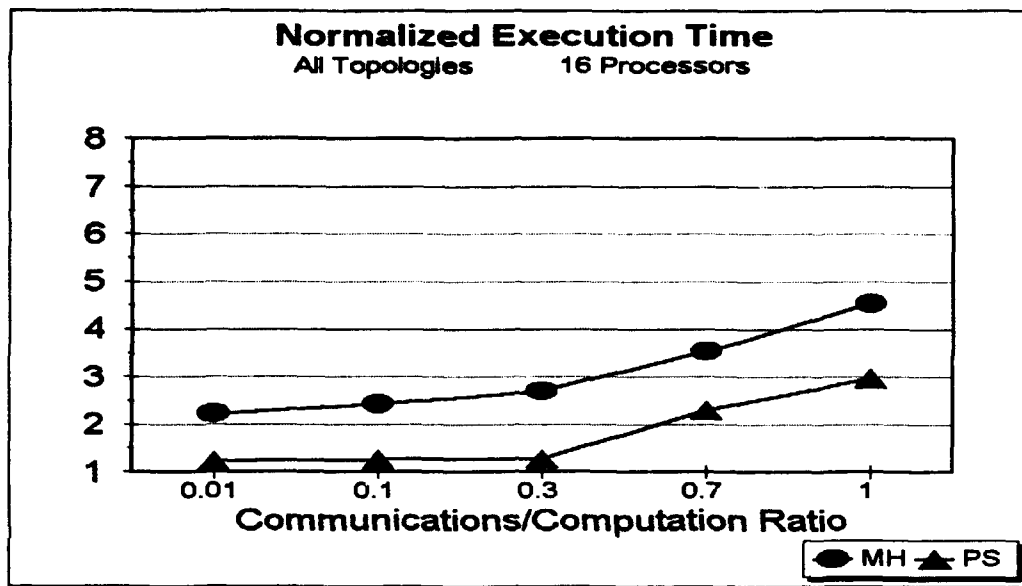


Figure 23: Normalized execution time across all topologies when communications/computation is varied.

3. Buffering Requirements.

As described earlier, due to the overlapping effect of task graphs in periodic scheduling, PS requires the assignment of indices to tasks in order to determine to which instance of a task graph that particular task belongs. In this section we briefly analyze these iteration indices, with the objective of determining the average number of instances of a task graph required in order to execute one complete task graph.

Figure 24 shows the average number of required graph instances on sixteen processor systems. The fully connected network is compared with other topologies, which are presented as an average. Note that the number of live instances required for all topologies other than the fully connected network declines at communications/computation ratios beyond 0.3.

As the amount of communications between tasks grow, so grows the necessary delay between a task and its successor if they are assigned to different processors. If the finish time of a node plus communications time and resource contention is greater than the

start time of a successor, the successor must begin execution on a subsequent iteration of the graph. Hence, in general, more communications implies more graph instances and more indices. At some point, however, as the communications/computation ratio grows, the PS heuristic will tend to assign nodes with a predecessor/successor relationship to the same processor, resulting in a later execution time, as observed earlier. When PS assigns these nodes with a precedence relationship to the same processor, the need for assigning different indices to the nodes is no longer present. The two nodes may now execute within the same instance of the task graph. Consequently fewer indices are required.

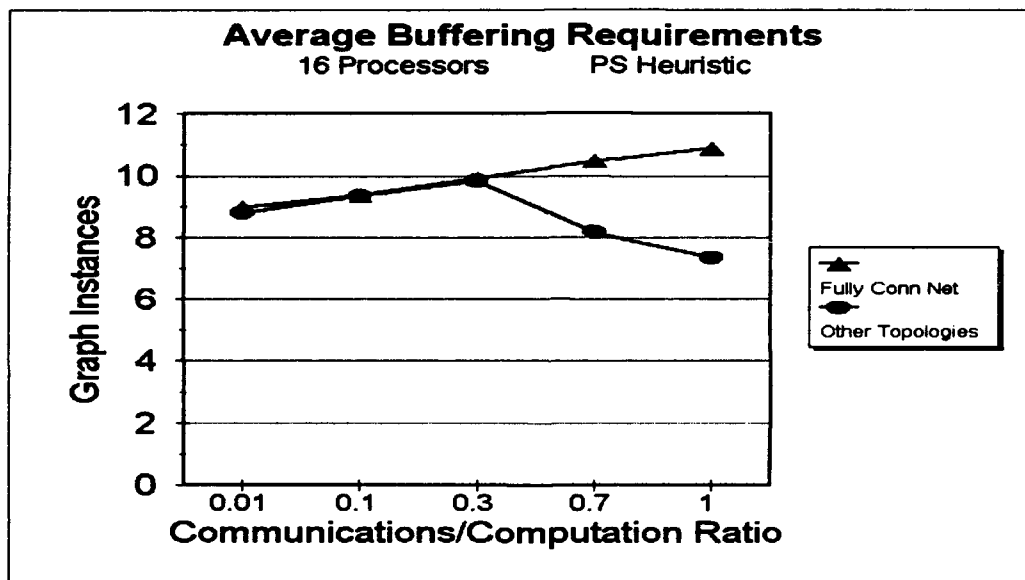


Figure 24: Average number of required graph instances on the ring and the fully connected network (FCN).

In Figure 25 this relationship between graph instances and execution time is graphically depicted. On topologies other than the fully connected network, the increase in execution time (due to tasks with a precedence relationship being mapped to the same processor) coincides with a decrease in the number of graph instances required.

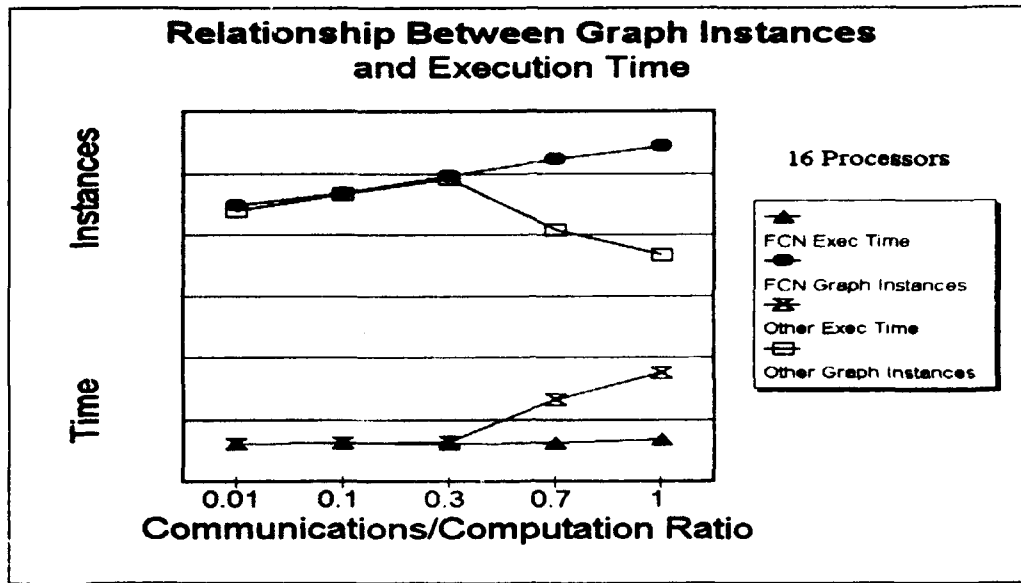


Figure 25: Relationship between graph instances and execution time at various communication/computation levels. (FCN is fully connected network).

C. GRAPH UNROLLING

An argument may be made that the actual throughput of MH can be better ascertained by unrolling. Unrolling consists of duplicating the task graph and presenting the multiple copies of the task graph for mapping by the heuristic. If i instances of a task graph are presented to MH for mapping, the total execution time of those instances will be less than i times the execution time of one instance of the graph. This is true since one instance of a task graph will have no precedence relationships with any other instance of a graph, and MH may map nodes of one instance without regard for nodes of another instance. For this reason one instance of a graph may be begun prior to the completion of a prior instance. This results in an average execution time of multiple instances being less than the execution time of one instance of the graph.

For the purposes of graph unrolling, we identify the unrolling factor u . The unrolling factor reflects the number of instances of a graph presented for mapping (i.e. if $u = 2$, two graph instances are mapped).

1. MH With Graph Unrolling

In this experiment, we consider the results of mapping MH to the sixteen processor ring and the sixteen processor fully connected network. Graphs will be mapped using MH for $u=2$, $u=3$ and $u=5$. These results are compared with the results of mapping using the PS heuristic without graph unrolling ($u=1$). All communications/computation ratios are considered.

Figure 26 shows the results of the mapping produced by MH with $u=2$ versus the mapping produced by PS with $u=1$. As seen in Figure 26, MH, on a fully connected network produces a mapping somewhat competitive with PS particularly at lower levels of communications/computation. On the ring and at higher levels of communications/computation, however, MH continues to lag behind PS.

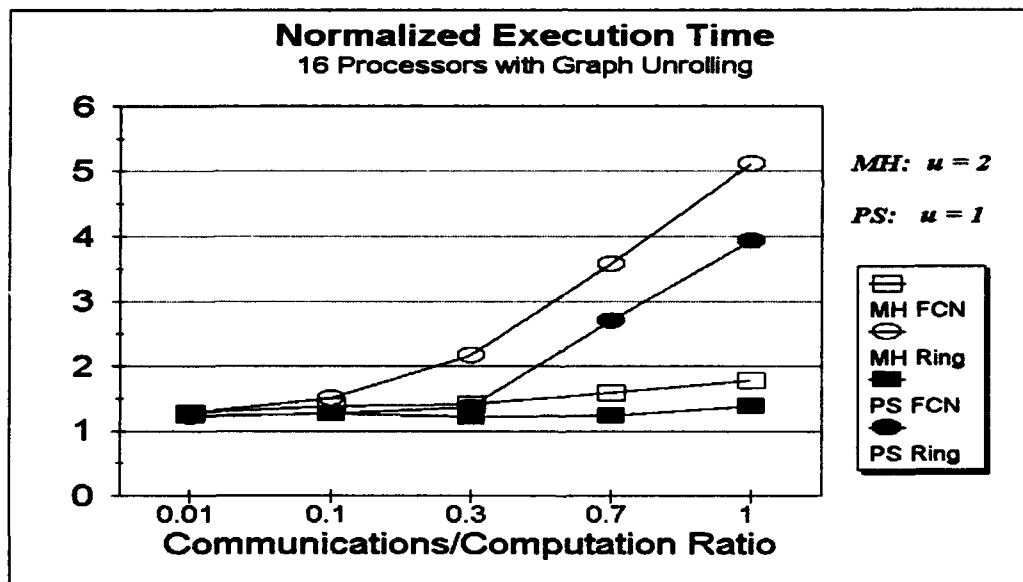


Figure 26: Comparison of PS ($u=1$) and MH ($u=2$).

When $u=3$ for MH and $u=1$ for PS, at lower levels of communications/computation, MH executes an instance of the task graph faster than or similar to PS on all topologies, as seen in Figure 27. MH on the ring, indicative of that topology's restricted communications flow, begins to lag behind PS at only moderate levels of communications/

computation. MH on the fully connected network tracks closely with PS at all communications/computation ratios.

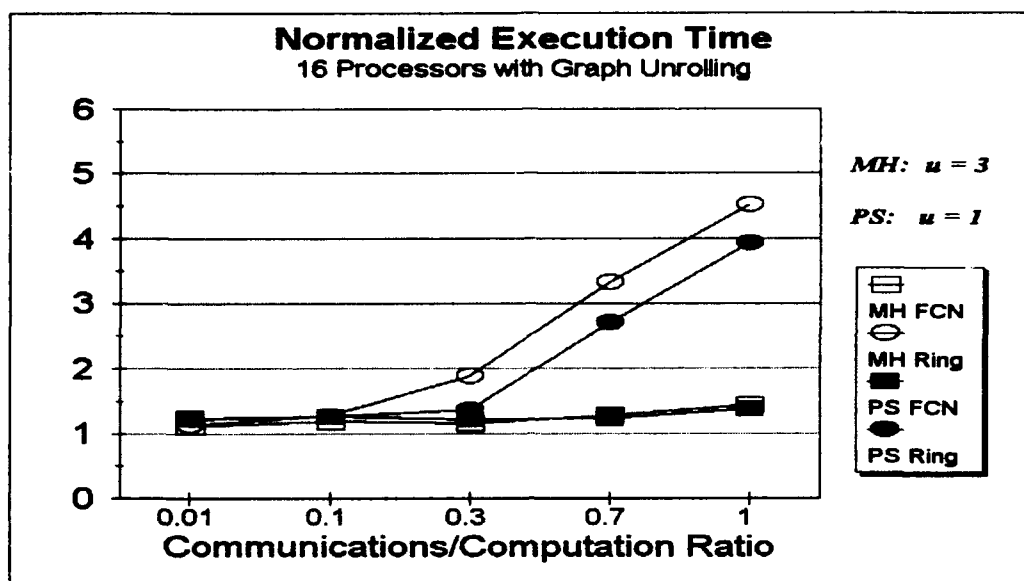


Figure 27: Comparison of PS ($u=1$) and MH ($u=3$).

Figure 28 shows that, with an unrolling factor of 5, the performance of MH has overtaken that of PS ($u=1$) in virtually every case. Furthermore, this superior performance is mostly independent of the communications/computation ratio. In the case of the ring of processors, MH ($u=5$) outperforms PS ($u=1$) at the lower levels of communications/computation. This shows that, in the long run, the average execution time of MH will be better than the execution time of one iteration of PS in most cases. However, MH is extremely sensitive to communications/computation increases, particularly on topologies with more restrictive communications flow, such as the ring.

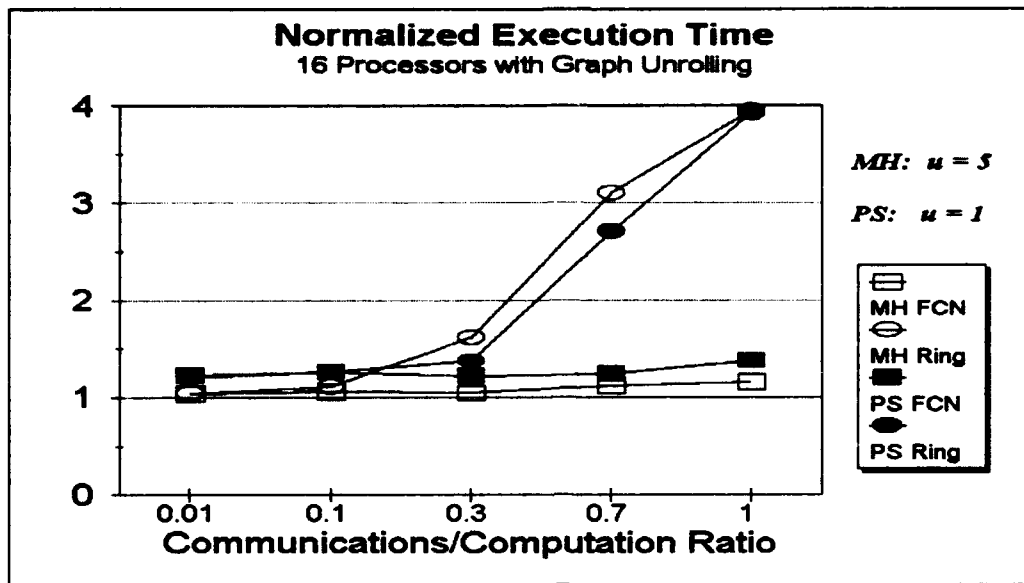


Figure 28: Comparison of PS ($u=1$) and MH ($u=5$).

2. PS With Graph Unrolling

Mapping multiple instances of a task graph using MH can provide results close to or, usually, superior to those of PS without graph unrolling, especially at lower levels of communications/computation ratios. The benefits of graph unrolling can apply to PS also. To accurately depict the potential of both heuristics, and to give a more realistic view of throughput of an iterative task graph, PS should also be allowed to take advantage of the graph unrolling previously applied to MH.

In this section we examine the effects of mapping multiple instances of a task graph to both PS and MH. Mappings produced by PS using unrolling factors of $u=2$, $u=3$ and $u=5$ are examined and compared with mappings produced by MH with the same unrolling factors. The ring and the fully connected network are considered on systems of sixteen processors at all communications/computation levels.

Figure 29 shows the results of mapping using MH and PS with $u=2$ for both heuristics. When $u=2$, the line depicting the performance of PS on the fully connected network is virtually flat, with execution time close to the ideal at all levels of

communications/computation. The performance of MH with $u=2$ has been previously described. The performance of PS on the ring, as seen previously, begins to decline somewhat at the higher communications levels, but still maintains a fair advantage over MH.

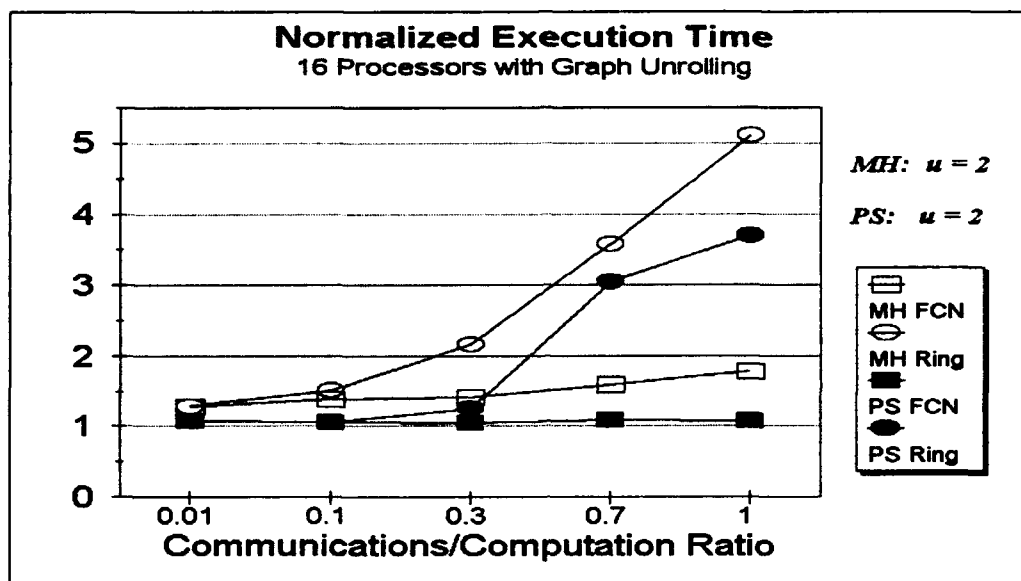


Figure 29: Comparison of PS ($u=2$) and MH ($u=2$).

Figure 30 compares the performance of PS and MH at $u=3$. Again, the performance of PS on the fully connected network approaches the ideal at all communications/computation ratios. MH produces mappings with normalized execution times competitive with those of PS at the lowest ratios. The disparity between the two grows, however, as the communications/computation ratios grow. In the case of the ring topology, the normalized execution times of MH track closely with those of PS at the lower communications levels.

Figure 31 depicts the performance of MH and PS when $u=5$ for both heuristics. Once again, PS on the fully connected network produces mappings which execute at levels generally close to the ideal given any communications/computation ratio. MH produces mappings which are fairly competitive with those produced by PS for both topologies

particularly at lower levels. PS continues to maintain an advantage, however, at the higher levels of communications/computation.

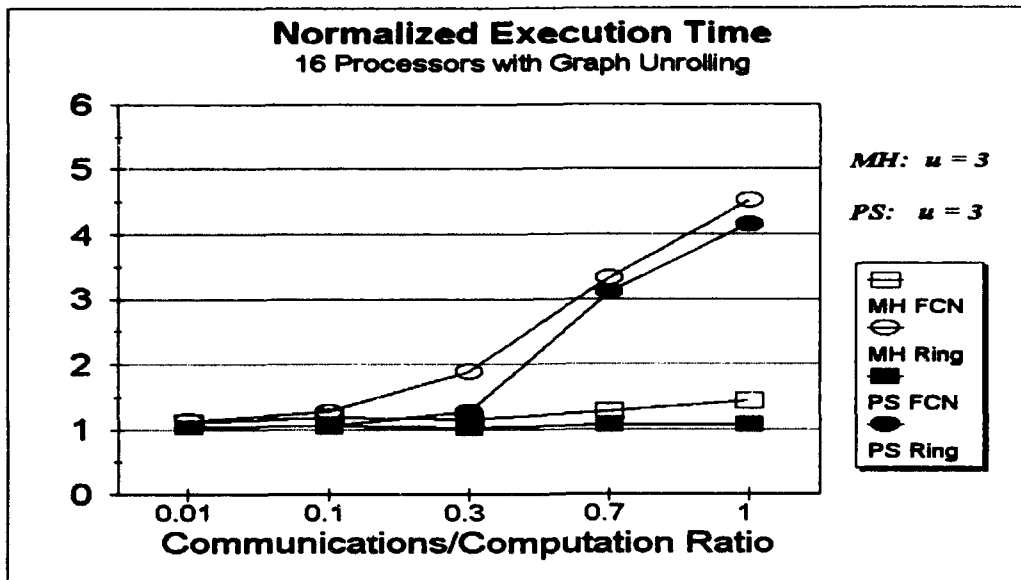


Figure 30: Comparison of PS ($u=3$) and MH ($u=3$).

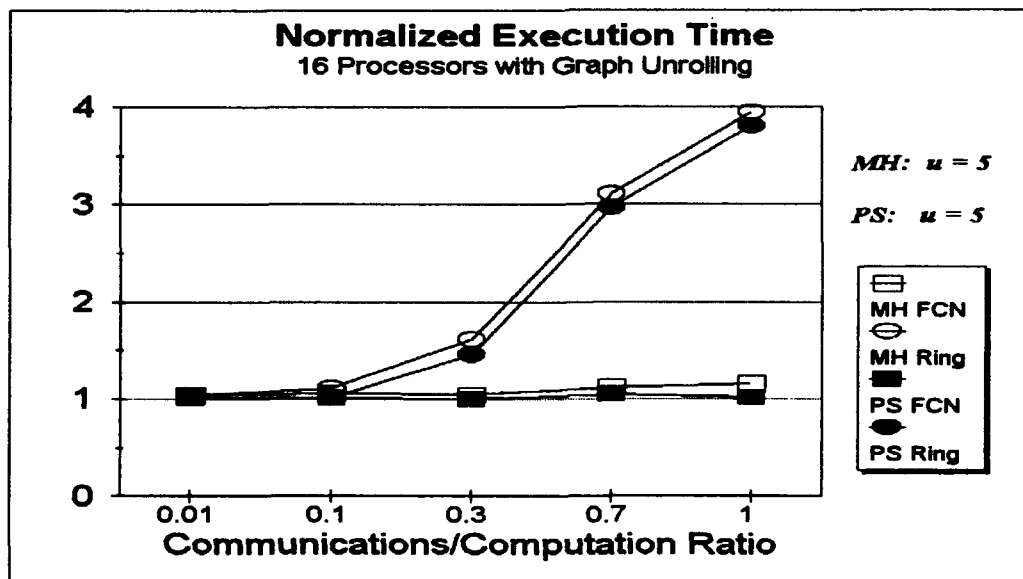


Figure 31: Comparison of PS ($u=5$) and MH ($u=5$).

V. CONCLUDING REMARKS

A. CONCLUSION

This thesis presents the Periodic Scheduling (PS) heuristic. The objective of PS is to maximize the throughput of an application represented by an iterative task graph when mapped to a distributed memory multiprocessor. This contrasts to the traditional objective of mapping heuristics, which has been minimization of execution time. The repetitive nature of the target applications makes possible the overlapping of distinct instances of their associated task graphs. PS capitalizes on this characteristic. In mapping tasks to processors, PS considers the topology of the multiprocessor system, communications between tasks, and contention for communications resources within the ICN.

The PS heuristic is described in the thesis in some detail. In tests, the performance of PS is compared to that of a version of the Mapping Heuristic (MH) which is extended for use in an iterative task environment.. It is shown by these tests that PS can provide superior throughput to the MH heuristic. PS is particularly superior in two cases. First, PS gives better throughput on systems composed of a larger number of processing elements. Second, the use of PS is advantageous when the target topology has a restricted pattern of communications flow, such as the ring of processors.

B. FUTURE WORK

Additional research using PS is needed in the area of mapping task graphs which contain cycles. Cycles within a task graph make additional pipeline stages problematical. The cycles become the bottleneck computations in the task graph, changing the dynamics of task execution and complicating mapping efforts.

A second area of potential research involves using PS in a two part scheduling strategy. Such a strategy would begin with an algorithm to cluster the nodes according to some heuristic. Once clustered, the nodes would be mapped to processors by PS.

More realistic communications models need to be applied to PS. Such issues as message initiation costs will no doubt affect the performance of PS and should be investigated.

LIST OF REFERENCES

- [AKI 93] Akin, C., *Efficient Scheduling of Real-Time Compute-Intensive Periodic Graphs on Large Grain Data Flow Multiprocessors*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1993.
- [BEL 92] Bell, H. A., *A Compile-Time Approach for Chaining and Execution Control in the AN/UYS-2 Parallel Signal Processor*, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1992.
- [BOK1 81] Bokhari, S. H., "On the Mapping Problem," *IEEE Transactions on Computers*, v. C-30, pp. 207-214, March 1981.
- [BOK2 81] Bokhari, S. H., "A Shortest Tree Algorithm for Optimal Assignments Across Space and Time in a Distributed Processor System," *IEEE Transactions on Software Engineering*, v. SE-7, pp. 583-589, November 1981.
- [BOK 88] Bokhari, S. H., "Partitioning Problems in Parallel, Pipelined and Distributed Computing," *IEEE Transactions on Computers*, v. 37, pp. 48-57, January 1988.
- [CAS 88] Casavant, T. L., and Kuhl, J. G., "A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems," *IEEE Transactions on Software Engineering*, v. 14, pp. 141-154, February 1988.
- [CHA 93] Chaudhary, V., and Aggarwal, J. K., "A Generalized Scheme for Mapping Parallel Algorithms," *IEEE Transactions on Parallel and Distributed Systems*, v. 4, pp. 328-345, March 1993.
- [ELR 90] El-Rewini, H., and Lewis, T. G., "Scheduling Parallel Program Tasks onto Arbitrary Target Machines," *Journal of Parallel and Distributed Computing*, v. 9, pp. 138-153, June 1990.
- [EVA 92] Evans, J. D., and Kessler, R. R., "A Communication-Ordered Task Graph Allocation Algorithm," Technical Report, University of Utah, Salt Lake City, Utah, April 1992.
- [GER 93] Gerasoulis, A., and Yang, T., "On the Granularity and Clustering of Directed Acyclic Task Graphs," *IEEE Transactions on Parallel and Distributed Systems*, v. 4, pp. 686-701, June 1993.

- [HAM 92] Hammond, S. W., *Mapping Unstructured Grid Computations to Massively Parallel Computers*, Doctoral Thesis, Rensselaer Polytechnic Institute, Troy, New York, February 1992.
- [HOA 93] Hoang, P. D., and Rabaey, J. M., "Scheduling of DSP Programs onto Multiprocessors for Maximum Throughput," *IEEE Transactions on Signal Processing*, v. 41, pp. 2225-2235, June 1993.
- [KER 70] Kernighan, B. W., and Lin, S., "An Efficient Heuristic Procedure for Partitioning Graphs," *The Bell System Technical Journal*, v. 49, pp. 291-307, February 1970.
- [LIT 91] Little, B. S., *A Technique for Predictable Real-Time Execution in the AN/UY-2 Parallel Signal Processing Architecture*, Master's Thesis, Naval Postgraduate School, Monterey, California, December, 1991.
- [LOV 88] Lo, V. M., "Heuristic Algorithms for Task Assignment in Distributed Systems," *IEEE Transactions on Computers*, v. 37, pp. 1384-1397, November 1988.
- [PEN 93] Peng, D., and Shin, K. G., "Optimal Scheduling of Cooperative Tasks in a Distributed System Using an Enumerative Method," *IEEE Transactions on Software Engineering*, v. 19, pp. 253-267, March 1993.
- [SHE 85] Shen, C., and Tsai, W., "A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems Using a Minimax Criterion," *IEEE Transactions on Computers*, v. C-34, pp. 197-203, March 1985.
- [SHU 92] Shukla, S., Little, B., and Zaky, A., "A Compile-Time Technique for Controlling Real-time Execution of Task-level Data-flow Graphs," *Proceedings of the 1992 International Conference on Parallel Processing*, v. II, pp. 49-56, August 1992.
- [SIH 93] Sih, G. C., and Lee, E. A., "Declustering: A New Multiprocessor Scheduling Technique," *IEEE Transactions on Parallel and Distributed Systems*, v. 4, pp. 625-637, June 1993.
- [STO 78] Stone, H. S., and Bokhari, S. H., "Control of Distributed Processes," *Computer*, v. 11, pp. 97-106, July 1978.
- [ULL 75] Ullman, J., "NP-Complete Scheduling Problems," *Journal of Computer and System Sciences*, v. 10, pp. 384-393, 1975.
- [ZHO 93] Zhou, H., *Distributed Computing of Weak and Strong Precedence Constrained Problems*, Doctoral Dissertation, University of Zurich, Zurich, Switzerland, July 1993.

INITIAL DISTRIBUTION LIST

- | | | |
|----|---|---|
| 1. | Defense Technical Information Center
Cameron Station
Alexandria, VA 22304-6145 | 2 |
| 2. | Dudley Knox Library
Code 52
Naval Postgraduate School
Monterey, CA 93943-5002 | 2 |
| 3. | Dr. Ted Lewis
Code CS, Computer Science Department
Naval Postgraduate School
Monterey, CA 93943-5000 | 1 |
| 4. | Dr. Amr M. Zaky
Code CS/Za
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943-5000 | 2 |
| 5. | Dr. Shridhar B. Shukla
Code EC/Sh
Electrical and Computer Engineering Department
Naval Postgraduate School
Monterey, CA 93943-5000 | 1 |
| 6. | LCDR Vinnie Squitieri
SPAWAR 231-Bh
Washington, DC 20363-5100 | 1 |
| 7. | Mr. David Kaplan
NRL (Code 5583)
Washington, DC 20375-5000 | 1 |
| 8. | LCDR Charles D. Kasinger, USN
Naval Security Group Activity Norfolk
1802 Powhatan
Norfolk, VA 23511-3379 | 2 |